CSc 120
Introduction to Computer Programing II

Adapted from slides by
Dr. Saumya Debray

01-d: Python review

python review: tuples

Tuples

```
>>>
>>> x = (111, 222, 333, 444, 555)
>>> x
(111, 222, 333, 444, 555)
>>> x[0]
111
>>> x[2]
333
>>> x[-1]
555
>>> x[-2]
444
>>>
```

a tuple is a sequence of values (like lists)

Tuples

```
>>>
>>> x = (111, 222, 333, 444, 555)
>>> x
(111, 222, 333, 444, 555)
>>> x[0]
111
>>> x[2]
333
>>> x[-1]
555
>>> x[-2]
444
>>>
```

a tuple is a sequence of values (like lists)

tuples use parens ()

- by contrast, lists use square brackets [ ]
  - parens can be omitted if no confusion is possible
- special cases for tuples:
  - empty tuple: ()
  - single-element tuple: must have comma after the element:

    (111,)

Tuples

```
>>>
>>> x = (111, 222, 333, 444, 555)
>>> x
(111, 222, 333, 444, 555)
>>> x[0]
111
>>> x[2]
333
>>> x[-1]
555
>>> x[-2]
444
>>>
```

a tuple is a sequence of values (like lists)

tuples use parens ()

- by contrast, lists use square brackets [ ]
  - parens can be omitted if no confusion is possible
- special cases for tuples:
  - empty tuple: ()
  - single-element tuple: must have comma after the element:

    (111,)

indexing in tuples works similarly to strings and lists

Tuples

```
>>> x = (111, 222, 333, 444, 555)
>>>
len(x)
5
>>> x[2:]
(333, 444, 555)
>>>
>>> x[:4]
(111, 222, 333, 444)
>>> x[1:4]
(222, 333, 444)
>>>
>>>
```

computing a length of a tuple:
similar to strings and lists

Tuples

```
>>> x = (111, 222, 333, 444, 555)
>>>
len(x)
5
>>> x[2:]
(333, 444, 555)
>>>
>>> x[:4]
(111, 222, 333, 444)
>>> x[1:4]
(222, 333, 444)
>>>
>>>
```

computing a length of a tuple:
similar to strings and lists

computing slices of a tuple:
similar to strings and lists

Tuples

```
>>> x = (111, 222, 333, 444, 555)
>>> x
(111, 222, 333, 444, 555)
>>>
>>> y =  (666, 777, 888)
>>>
>>> x + y
(111, 222, 333, 444, 555, 666, 777, 888)
>>>
>>> y * 3
(666, 777, 888, 666, 777, 888, 666, 777, 888)
>>>
```

+ and * work similarly on tuples as for lists and strings

Tuples

```
>>> x = (111, 222, 333, 444, 555)
>>> for item in x:
        print(item)

111
222
333
444
555
>>>
>>> 222 in x
True
>>> 999 in x
False
>>>
```

iterating through the elements of a tuple: similar to lists and strings

Tuples

```
>>> x = (111, 222, 333, 444, 555)
>>>  for item in x:
        print(item)


111
222
333
444
555
>>>
>>> 222 in x
True
>>> 999 in x
False
>>>
```

iterating through the elements of a tuple: similar to lists and strings

checking membership in a tuple: similar to lists and strings

Tuples

```
>>> x = (111, 222, 333, 444, 555)
>>> x
(111, 222, 333, 444, 555)
>> x[2]
333
>>>
>>> x[2] = 999
Traceback (most recent call last):
  File "<pyshell#102>", line 1, in <module>
    x[2] = 999
TypeError: 'tuple' object does not support item assignment
>>>
```

tuples are not mutable

Sequence types: mutability

tuples are immutable

```
>>> x = ( ['aa', 'bb'], ['cc', 'dd'], ['ee'] )
>>> x[0] = 'ff'
Traceback (most recent call last):
  File "<pyshell#108>", line 1, in <module>
    x[0] = 'ff'
TypeError: 'tuple' object does not support item assignment
```

Sequence types: mutability

```
>>> x = ( ['aa', 'bb'], ['cc', 'dd'], ['ee'] )
>>> x[0] = 'ff'
Traceback (most recent call last):
  File "<pyshell#108>", line 1, in <module>
    x[0] = 'ff'
TypeError: 'tuple' object does not support item assignment
>>> x[0][0] = 'ff'
>>> x
(['ff', 'bb'], ['cc', 'dd'], ['ee'])
```

tuples are immutable

lists are mutable

Sequence types: mutability

```
>>> x = ( ['aa', 'bb'], ['cc', 'dd'], ['ee'] )
>>> x[0] = 'ff'
Traceback (most recent call last):
  File "<pyshell#108>", line 1, in <module>
    x[0] = 'ff'
TypeError: 'tuple' object does not support item assignment
>>> x[0][0] = 'ff'
>>> x
(['ff', 'bb'], ['cc', 'dd'], ['ee'])
>>> x[0][0][0] = 'a'
Traceback (most recent call last):
  File "<pyshell#112>", line 1, in <module>
    x[0][0][0] = 'a'
TypeError: 'str' object does not support item assignment
>>>
```

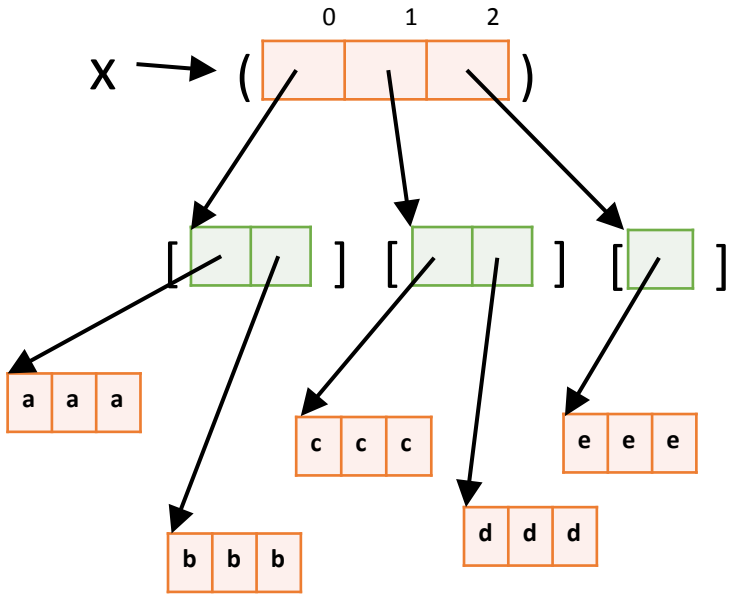
tuples are immutable

lists are mutable

strings are immutable

Sequence types: mutability

```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more information.
>>> x = ( ['aaa', 'bbb'], ['ccc', 'ddd'], ['eee'])
>>>
>>> x[0] = 'fff'
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    x[0] = 'fff'
TypeError: 'tuple' object does not support item assignment
>>>
>>> x[0][0] = 'fff'
>>> x
(['fff', 'bbb'], ['ccc', 'ddd'], ['eee'])
>>>
>>> x[0][0][0] = 'a'
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    x[0][0][0] = 'a'
TypeError: 'str' object does not support item assignment
>>>
```



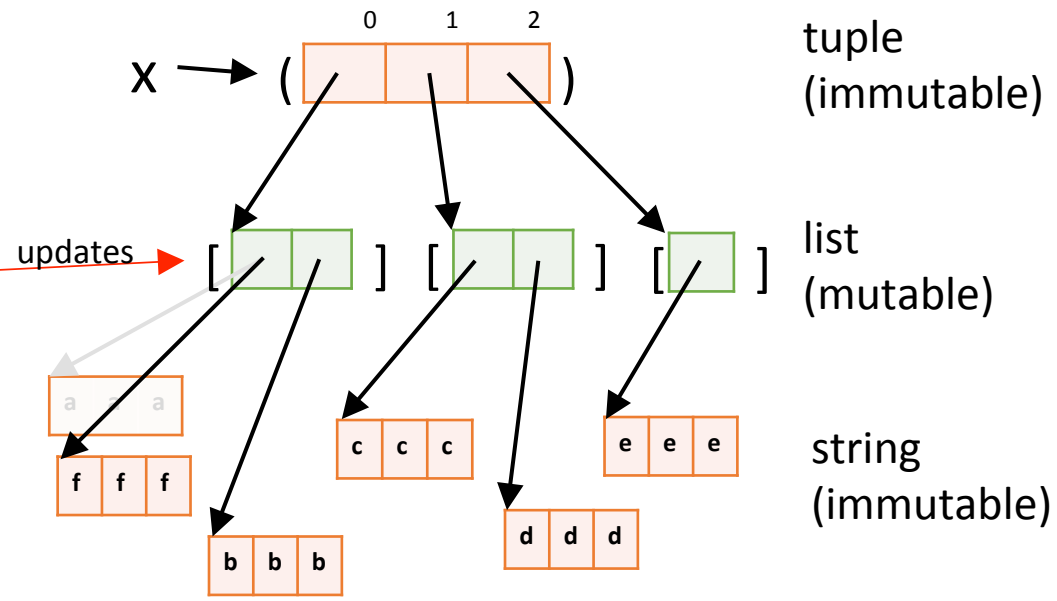
tuple
(immutable)

list
(mutable)

string
(immutable)

Sequence types: mutability

```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> x = ( ['aaa', 'bbb'], ['ccc', 'ddd'], ['eee'])
>>>
>>> x[0] = 'fff'
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    x[0] = 'fff'
TypeError: 'tuple' object does not support item assignment
>>>
>>> x[0][0] = 'fff'
>>> x
(['fff', 'bbb'], ['ccc', 'ddd'], ['eee'])
>>>
>>> x[0][0][0] = 'a'
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    x[0][0][0] = 'a'
TypeError: 'str' object does not support item assignment
>>>
```



EXERCISE

```
>>> x = [ (1, 2, 3), (4, 5, 6), (7, 8, 9) ]
>>> x[0][0] = (2, 3, 4)
```

what do you think will be printed out?

```
>>> x[0] = [ 2, 3, 4 ]
```

what do you think will be printed out?

Why use tuples?

At the implementation level, tuples are much simpler than lists:

- lists are mutable; tuples are immutable
  - this means that the implementation can process tuples without having to worry about the possibility of updates
- lists have methods (e.g., append); tuples do not have methods

$\Rightarrow$ Tuples can be implemented more efficiently than lists

Summary: sequence types

Sequence types include: strings, lists, and tuples

Operation	Result
`x in s`	`True` if an item of *s* is equal to *x*, else `False`
`x not in s`	`False` if an item of *s* is equal to *x*, else `True`
`s + t`	the concatenation of *s* and *t*
`s * n` or `n * s`	equivalent to adding *s* to itself *n* times
`s[i]`	*i*th item of *s*, origin 0
`s[i:j]`	slice of *s* from *i* to *j*
`s[i:j:k]`	slice of *s* from *i* to *j* with step *k*
`len(s)`	length of *s*
`min(s)`	smallest item of *s*
`max(s)`	largest item of *s*
`s.index(x[, i[, j]])`	index of the first occurrence of *x* in *s* (at or after index *i* and before index *j*)
`s.count(x)`	total number of occurrences of *x* in *s*

The elements are: *i, i+k, i +2k, ...*

Source: https://docs.python.org/3/library/stdtypes.html#sequence-types-list-tuple-range

python review: dictionaries

Dictionaries

- A dictionary is like an array, but it can be indexed using strings (or numbers, or tuples, or any immutable type)
  - the values used as indexes for a particular dictionary are called its *keys*
  - think of a dictionary as an unordered collection of *key* : *value* pairs
  - empty dictionary: {}
- It is an error to index into a dictionary using a non-existent key

Dictionaries

```
>>> crs_units = {}
>>> crs_units['csc 110'] = 4
>>> crs_units['csc 120'] = 4
>>> crs_units['csc 352'] = 3
>>> course = 'csc 110'
>>>
>>> crs_units[course]
4
>>> crs_units
{'csc 110': 4, 'csc 120': 4, 'csc 352': 3}
>>>
```

empty dictionary

Dictionaries

```
>>> crs_units = {}
>>> crs_units['csc 110'] = 4
>>> crs_units['csc 120'] = 4
>>> crs_units['csc 352'] = 3
>>> course = 'csc 110'
>>>
>>> crs_units[course]
4
>>> crs_units
{'csc 110': 4, 'csc 120': 4, 'csc 352': 3}
>>>
```

empty dictionary

populating the dictionary

- in this example, one item at a time

Dictionaries

```
>>> crs_units = {}
>>> crs_units['csc 110'] = 4
>>> crs_units['csc 120'] = 4
>>> crs_units['csc 352'] = 3
>>> course = 'csc 110'
>>>
>>> crs_units[course]
4
>>> crs_units
{'csc 110': 4, 'csc 120': 4, 'csc 352': 3}
>>>
```

empty dictionary

populating the dictionary

- in this example, one item at a time

looking using keys (indexing)

Dictionaries

```
>>> crs_units = {}
>>> crs_units['csc 110'] = 4
>>> crs_units['csc 120'] = 4
>>> crs_units['csc 352'] = 3
>>> course = 'csc 110'
>>>
>>> crs_units[course]
4
>>> crs_units
{'csc 110': 4, 'csc 120': 4, 'csc 352': 3}
>>>
```

empty dictionary

populating the dictionary

- in this example, one item at a time

looking using keys (indexing)

- we can populate it using this syntax

Dictionaries

```
>>> crs_units = {}
>>> crs_units['csc 110'] = 4
>>> crs_units['csc 120'] = 4
>>> crs_units['csc 352'] = 3
>>> course = 'csc 110'
>>>
>>> crs_units[course]
4
>>> crs_units
{'csc 110': 4, 'csc 120': 4, 'csc 352': 3}
>>> >> crs_units['mis 115']
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    crs_units['mis 115']
KeyError: 'mis 115'
>>>
```

empty dictionary

populating the dictionary

- in this example, one item at a time

looking using keys (indexing)

- we can populate it using this syntax

indexing with a key not in the dictionary is an error (**`KeyError`**)

Dictionaries

```
>>> crs_units = {}
>>> crs_units['csc 110'] = 4
>>> crs_units['csc 120'] = 4
>>> crs_units['csc 352'] = 3
>>> course = 'csc 110'
>>>
>>> crs_units
{'csc 110': 4, 'csc 120': 4, 'csc 352': 3}
>>> 'mis 115' in crs_units
False
>>>
```

indexing with a key not in the dictionary is an error (**`KeyError`**)

in operator:

- returns *True* if a key is in the dictionary an *False* otherwise

Dictionaries

```
>>>
>>> crs_units = { 'csc 110': 4, 'csc 120': 4, 'csc 352': 3}
>>>
>>> crs_units['csc 110']
4
>>>
>>> crs_units.keys()
dict_keys(['csc 110', 'csc 120', 'csc 352'])
>>>
>>> crs_units.values()
dict_values([4, 4, 3])
>>>
```

initializing the dictionary

- in this example, several items at once

Dictionaries

```
>>>
>>> crs_units = { 'csc 110': 4, 'csc 120': 4, 'csc 352': 3}
>>>
>>> crs_units['csc 110']
4
>>>
>>> crs_units.keys()
dict_keys(['csc 110', 'csc 120', 'csc 352'])
>>>
>>> crs_units.values()
dict_values([4, 4, 3])
>>>
```

methods for getting:

- the keys
- the values

Dictionaries

```
>>>
>>> crs_units = { 'csc 110': 4, 'csc 120': 4, 'csc 352': 3}
>>>
>>> crs_unist['csc 110']
4
>>>
>>> crs_units.keys()
dict_keys(['csc 110', 'csc 120', 'csc 352'])
>>>
>>> list(crs_units.keys())
['csc 110', 'csc 120', 'csc 352']
>>>
>>>
```

get a list of the keys

Dictionaries

```
>>> crs_units = { 'csc 110': 4, 'csc 120': 4, 'csc 352': 3}
>>>
>>> for crs in crs_units:
        print(crs, crs_units[crs])

csc 110 4
csc 352 3
csc 120 4
>>>
```

We can use a **for** loop to iterate through a dictionary

Dictionaries

```
>>> crs_units = { 'csc 110': 4, 'csc 120': 4, 'csc 352': 3}
>>>
>>> for crs in crs_units:
        print(crs  +  ":",  crs_units[crs], " units")
```

We can use a **for** loop to iterate through a dictionary

```
csc 110 : 4 units
csc 352 : 3 units
csc 120 : 4 units
>>>
```

Note: this may not list the items in the dictionary in the same order as when they were inserted

Dictionaries

```
>>> crs_units = { 'csc 110': 4, 'csc 120': 4, 'csc 352': 3}
>>>
>>> for crs in crs_units:
        # print(crs +  ":", crs_units[crs], "units")
        print("{0}:{1} units".format(crs, crs_units[crs]))


csc 110: 4 units
csc 352: 3 units
csc 120: 4 units
>>>
```

format

EXERCISE

```
>>> crs_units = { 'csc 352' : 3, 'csc 120': 4, 'csc 110': 4 }
>>> for crs in 
        print( "{0} : {1} units".format( crs, crs_units[crs] )
```

```
csc 110 : 4 units
csc 120 : 4 units
csc 352 : 3 units
>>>
```

How can we get the dictionary contents to be printed out in sorted order of the keys?
(I.e., what goes in the box?)

EXERCISE

Write a function count_chars(s) that takes a string s and returns a dictionary of the counts of all characters in the string.

EXERCISE

Write a function count_chars(s) that takes a string s and returns a dictionary of the counts of all characters in the string.

```
def count_chars(s):
    counts = {}
    s = s.lower()
    for c in s:
        if c in counts:          #if we have seen c, increment its count
            counts[c] = counts[c] +  1
        else:                    #otherwise, it is the first occurrence
            counts[c] = 1
    return counts
```

Dictionary Summary

Operation	Result
dict()	Return an empty dictionary.
len(d)	Return the number of items in the dictionary `d`.
d[key]	Return the item of d with key key. Raises an error if key is not in the dictionary.
d[key] = value	Set `d[key]` to value.
del d[key]	Remove `d[key]` from `d`. Raises an error if key is not in the dictionary
key in d	Return `True` in `d` has a key key, else `False`.
key not in d	Equivalent to not key in `d`.
keys()	Return the dictionary's keys.
values()	Return the dictionary's values.
items()	Return the dictionary's items as tuples.